1.label的作用：

总结:

FOR属性

功能：表示Label标签要绑定的HTML元素，你点击这个标签的时候，所绑定的元素将获取焦点。

用法：

```
<label FOR="InputBox">姓名</label>
<input ID="InputBox" type="text"/>
```

ACCESSKEY属性：

功能：表示访问Label标签所绑定的元素的热键，当您按下热键，所绑定的元素将获取焦点。

用法：

```
<Label FOR="InputBox" ACCESSKEY="N">姓名</Label>
<input ID="InputBox" tabindex="N" type="text"/>
```

局限性：accessKey属性所设置的快捷键不能与浏览器的快捷键冲突，否则将优先激活浏览器的快捷键。

2.凹凸文字和空心文字

```
    <style>
        body {
            background-color: grey;
            font: bold 6em "micro";
        }
        div {
            margin: 30px;
            /*将背景颜色与字体颜色调成一样，再给文字设置两个相反方向的阴影*/
            color: grey;
        }
        .to {
            text-shadow: -1px -1px 1px white, 1px 1px 1px #000;
        }/*白色衬黑色*/

        .ao {
            text-shadow: -1px -1px 1px #000, 1px 1px 1px white;
        }/*黑色衬托白色*/
```

```
18
19
20
21          p {
22              font-size: 50px;
23              /*伪空心元素，与背景色有关*/
24              -webkit-text-fill-color: white;/*文字空心内部填充颜色*/
25              -webkit-text-stroke-color: blue;/*文字镂空边框颜色*/
26              /*边框粗细*/
27              -webkit-text-stroke-width: 2px;
28              /*与上一条可合写为-webkit-text-stroke:2px blue;*/
29          }
30      </style>
31  </head>
32
33  <body>
34      <div class="to">我是凸起的文字</div>
35      <div class="ao">我是凹下的文字</div>
36      <p>我是空心的文字，酷不酷？</p>
37  </body>
```

3.盒子模型

组成：内容、边框、内边距、外边距，页面中默认加载出来的盒子为标准模型

分类：标准盒模型，IE盒模型

IE盒模型（怪异盒模型）：(content+padding+border)(height+width)+margin

外边距塌陷：

1.嵌套的两个盒子：

解决方法：

给父级：overflow：hidden；

2.相邻的两个盒子

4.动画animation

```
1  <style>
2      *{
3          margin: 0;
4          padding: 0;
5      }
```

```
    .box1{
        width: 300px;
        height: 200px;
        background-color: #ccc;
        animation: animate1 2s infinite;
    }
    /* @keyframes animate1 {
        from{
            transform: translate(0,0);
        }to{
            transform: translate(-300px,0);
        }
    } */
    @keyframes animate1 {
        0%{
            transform: translate(0,0);
        }
        50%{
            transform: translate(-300px,0);
        }
        100%{
            transform: translate(0,0);
        }
    }
  </style>
</head>
<body>
    <div class="box1">box1</div>
</body>
```

5.多媒体查询

```
    <style>
        * {
            margin: 0;
            padding: 0;
        }

```

```
        .bigBox {
            height: 400px;
            font-size: 50px;
            background-color: #ccc;
        }

        /* 1000px-1300px font-size: 30px;  background-color: blue;*/
        @media screen and (min-width:1000px) and (max-width:1300px) {
            .bigBox {
                font-size: 30px;
                background-color: blue;
            }
        }
        /* <1000px  font-size: 20px;  background-color: red;*/
        @media screen and (max-width:1000px) {
            .bigBox {
                font-size: 20px;
                background-color: red;
            }
        }
    </style>
</head>

<body>
    <div class="bigBox">bigBox</div>
</body>
```

6.移动端布局

```
  <style>
    *{
        margin: 0;
        padding: 0;
    }
    html{
        /* font-size: 50px; */
        /*
            750px   375px    100vw
```

```
            100px    50px      ?
        */
        font-size: calc((100*100vw) / 750);
    }
    .box{
        font-size: 2rem;
        background-color: #ccc;
    }

  </style>
</head>
<body>
    <!--
        vw:
        vh:
        rem: 默认 1rem = 16px
     -->
    <div class="box">box</div>
</body>
```

8.写页面步骤

```
<body>
    <style>
        /* 当屏幕宽度小于1400px */
        /* @media screen and (max-width:1400px) {
            .bigBox{
                width: 1400px;
            }
        } */
    </style>
    <!-- 整个页面左右没有间距 -->
    <div class="bigBox">
        <!-- 一行一行的实现，每一行用一个div -->
        <!-- 第一行: -->
        <div class="header"></div>
        <!-- 第二行:导航 -->
        <div class="nav"></div>
        <!-- 第三行 -->
```

```
        <div class="con1"></div>
        <!-- ... -->
    </div>

    <style>
        *{
            margin: 0;
            padding: 0;
        }
        .header{
            height: 80px;
            background-color: #ccc;
        }
        .navBox{
            height: 600px;
            background-color: #f00;
        }
        .content{
            height: 1200px;
            background-color: green;

            width: 1200px;
            margin: 0 auto;
            margin-top: -200px;
        }
        /* 当屏幕宽度小于1200px */
        @media screen and (max-width:1200px) {
            /* .header{
                width: 1200px;
            }
            .navBox{
                width: 1200px;
            } */
            .bigBox{
                width: 1200px;
            }
        }

        .nav{
            height: 60px;
```

```
            background-color: rgba(0,0,0, 0.7);
            color: white;
            display: flex;
            justify-content: space-between;
            align-items: center;
        }
        .nav-left{
            margin-left: 80px;
        }
        .nav-center{
            margin-right: 50px;
        }
        .nav-right{
            margin-right: 80px;
        }
    </style>

    <!-- 整个页面大部分行的内容左右有间距 -->
    <div class="bigBox">
        <!--
            1.先对整个页面进行大的区块划分，每一块用一个div
            2.每一大块每一大块的实现（每一大块里面可能会存在很多行）
         -->
        <!-- 王者荣耀 -->
        <!-- 顶部第一大块 -->
        <div class="header"></div>
        <!-- 第二大块 -->
        <div class="navBox">
            <div class="nav">
                <div class="nav-left">nav-left</div>
                <div class="nav-center">nav-centernav-centernav-center</div>
                <div class="nav-right">nav-right</div>
            </div>
        </div>
        <!-- 第三大块 -->
        <div class="content">
            <div class="con1">con1</div>
            <div class="con2">con2</div>
        </div>
```

```
    </div>
    <!--
        每一行的实现步骤：
        1.先对当前行进行区块的划分，每一块用一个div
        2.每一块每一块的实现
        3.让每一小块都排成一行：用弹性布局实现，把这一行的div(每一小块的父亲)设为弹性容器
        4.用容器属性进行布局排列：justify-content和align-items
        5.用margin或者padding进行间距调整
        6.最后实现鼠标移入（相对定位和绝对定位）

        最后实现整个页面的固定定位的盒子

        注意点：能不给标签设置固定宽高，就不要设置固定宽高

        display: flex(独占一行的容器)|inline-flex（不独占一行的容器）;
     -->
</body>
```

9.页面跳转

```
<body>
    <button id="btn1">02-模态框</button>
    <button id="btn2">刷新页面</button>

    <!-- <a href="https://www.baidu.com/"></a> -->
    <script>
        // 跳转页面
        // window.location.href = "./02-模态框.html";
        document.getElementById("btn1").onclick = function(){
            // window.location.href = "./02-模态框.html";
            // window.location.replace("./02-模态框.html");
            // window.location.assign("./02-模态框.html");
            // window.open("./02-模态框.html");
            window.open("./02-模态框.html","_self");
        }
        document.getElementById("btn2").onclick = function(){
            // 刷新页面
            window.location.reload();
        }
```

```
        // 系统信息
        console.log(window.navigator);
    </script>
</body>
```

10.头像上传

```
    <style>
        .box{
            display: flex;
        }
        .headLabel{
            width: 120px;
            height: 160px;
            border: 2px solid #ccc;
            background-size: 100% 100%;
        }
    </style>
</head>
<body>
    <div class="box">
        头像：<label for="headinput" class="headLabel"></label>
        <!-- <input id="headinput" type="file" accept="image/*" multiple> -->
        <input id="headinput" type="file" accept="image/*">
    </div>
    <script>
        let headinput =  document.getElementById("headinput");
        let headLabel = document.querySelector('.headLabel');
        // 监听输入框内容变化的事件：change
        headinput.onchange = function(){
            // console.log("输入框内容一改变就执行这个事件处理函数");
            // 获取图片文件
            console.log(headinput.files);
            let file = headinput.files[0];
            // 获取图片编码
            let fs = new FileReader();
            fs.readAsDataURL(file);//读取文件  注意：文件读取是异步的
            // 文件读取结果  等到文件读取完成再获取读取结果
            fs.onload = function(){
```

```
                // 文件读取完成后才会执行
                console.log(fs.result);
                let r = fs.result;//图片文件编码读取结果
                // 展示图片
                headLabel.style.backgroundImage = `url(${r})`;
            }
        }

        // Function  Object  Array  String  Number...
    </script>
```

11.JS数据类型

```
<!--
        基本数据类型：string: es5: ""  ''    es6: `${}`
                    number  null  undefined  Boolean  Symbol  bigInt(100n)，栈
        引用数据类型：Array  Object  Function，堆
     -->
    <script>
        let num = 100n;
        console.log("afsf" - 23);
        console.log("========================对象=====================");
        // 对象的创建
        let obj1 = new Object();
        obj1.info = "obj1";

        let obj2 = { info: "obj2" };

        // 必须传入一个值：null 或者 对象，传入的是谁，那么创建出来的对象的原型就指向谁
        let obj3 = Object.create(null);
        obj3.info = "obj3";

        // 工厂模式
        function createObj() {
            let obj = new Object();
            return obj;
        }
        let obj4 = createObj();

```

```
        // 构造函数：构造函数的函数名首字母一般会大写，函数里面会出现this，这个this指向的是实例化对象，函数默认返回this
        function Person(name, age) {
            this.name = name;
            this.age = age;
        }
        let p1 = new Person("AAA", 21);

        console.log(obj1, obj2, obj3, obj4, p1);
        console.log(p1.age, p1["age"]);

        // 删除对象的属性
        delete obj1.info;
        console.log(obj1);
    </script>
```

12.sort方法

```
let myArr = ["function","object","Array","Number","String","Math"];
        // 根据首字母排序
        console.log(myArr.sort((a,b)=>{
            if(a.toLocaleLowerCase() < b.toLocaleLowerCase()){
                return -1
            }
        }));
```

方法封装：

```
    <script>
        let arr = [10, 4, 100, 66, 7];
        let sort_r = [...arr].sort((a, b) => {
            // return a-b;
            if (a < b) {
                return -1;
            }
        });
        console.log(sort_r, arr);

        Array.prototype.mysort = function (fn) {
```

```
12
13              // 逻辑代码
14              for (let i = 0; i < this.length; i++) {
15                  for (let j = i + 1; j < this.length; j++) {
16                      // if (this[i] > this[j]) {
17                      // 注意：负值转换为布尔值为true
18                      if ( fn(this[j],this[i]) < 0) {//小于0就调换位置
19                          [this[i], this[j]] = [this[j], this[i]];
20                      }
21                  }
22              }
23
24              return this;//返回结果
25          }
26          let mysort_r = arr.mysort((a, b) => {
27              // return a-b;
28              if (a < b) {
29                  return -1;
30              }
31
32              // return 1;
33          });
34          console.log(mysort_r);
35
36          // 从小到大
37          for (let i = 0; i < arr.length; i++) {
38              for (let j = i + 1; j < arr.length; j++) {
39                  if (arr[i] > arr[j]) {
40                      /* let middle = arr[i];
41                      arr[i] = arr[j];
42                      arr[j] = middle; */
43                      [arr[i], arr[j]] = [arr[j], arr[i]];
44                  }
45              }
46          }
47          console.log(arr);
48      </script>
```

13.函数、作用域、闭包

```
<!--
    作用域：作用域分为全局作用域和局部作用域
    全局作用域：在最顶层代码里面定义的变量就为全局作用域
    局部作用域：又叫函数作用域

    预编译：GO{

    }
    AO{

    }

    闭包：将函数内部和函数外部搭建起来的一个桥梁，使函数外部可以用到函数内部的变量
    优点：使函数外部用到函数内部的变量
    缺点：造成内存泄漏

    垃圾回收：
    1.引用计数
    2.标记清除
 -->
<script>
    /*
        GO{
            g1:
            num:
            gfn1:
        }
    */
    var g1 = 100;
    var num = 111;
    gfn1();
    function gfn1() {
        /*
            AO{
                num:
            }
        */
        console.log("gfn1");
        console.log(num);//undefined
```

```
            var num = 200
        }

        function gfn2() {
            /*
                AO{
                    num
                }
            */
            var num = 300
        }
        console.log("===============");
        function fn() {
            var n = 10;
            return function () {
                return ++n;
            }
        }
        let fn_r = fn();
        console.log(fn_r());// 11
        console.log(fn_r());// 12
        console.log(fn_r());// 13
        // fn_r = null;
    </script>
</body>
```

14.数据转换

```
<script>
        /*
            转Number :
            1.-  *  /  %
            2.parseInt()
            3.Number()
        */
        console.log(parseInt("20px"));//20
        console.log(new Number(123));
        console.log(Number("234adsf"));
        console.log(Number("321"));
```

```
        /*
            转字符串：值.toString();
                        String()
        */
        console.log(new String("a"));
        console.log(String({info: "对象"}));
        console.log(JSON.stringify({info: "对象"}));
        console.log(JSON.parse(`{"info": "对象"}`));
        /*
            转Boolean值：Boolean()
                        !!值
            false: undefined  null  0  ""  false  NaN
        */
        console.log(new Boolean(true));
    </script>
```

15.原型与原型链

```
<!--
原型：js里面一切皆可以看做对象，每个对象都有自己的原型，
实例化对象的原型__proto__指向的是构造函数的原型prototype，
构造函数的原型对象prototype 的原型__proto__ 指向的是 Object的原型prototype，
Object的原型prototype 的原型__proto__ 指向null。


原型链：在操作对象的时候，可以获取对象上面的某个属性，首先会在自身身上找这个属性，
没找到就去原型上面找，原型上面没找到就去原型的原型上面找，直到找到最后还没找到，
那么就返回undefined  如果是获取某个方法调用，最后没找到就会报错。
 -->
```

16.数组总结

```
<!--
        数组方法：
        增加：push  unshift
        删除：pop   shift
        删除、截取、替换：splice(start[,length,value1,value2...])
        截取：slice(start,end:不包含);
        查找：indexOf
```

```
        转字符串：join
        合并数组：concat
        遍历：forEach
        遍历数组，返回一个新数组：map
        排序：sort
        判断数组里面是否每一个元素都满足条件：every
        判断数组里面是否有一些元素都满足条件：some
        颠倒数组：reverse
        累加、累乘...：reduce
        过滤器：filter
        查找第一个满足条件的值：find
        转一维数组：flat
        填充：fill
    -->
```

17.reduce封装

```
let arr = [1, 2, 3, 4, 10, 5];

        Array.prototype.myReduce = function (fn, initial) {
            let result, i;
            if (!initial) {
                result = this[0];
                i = 1;
            } else {
                result = initial;
                i = 0;
            }
            for (; i < this.length; i++) {
                result = fn(result, this[i], i, this);
            }
            return result;
        };

        let arr2 = arr.myReduce((tital, val, ind, arr) => {
            console.log(tital, val, ind, arr);
            return tital * val;
        })
        console.log(arr2);
```

18.字符串方法

```
        split: 将字符串切割成数组

          字符串截取:
          substring(start,end:不包含)
          slice(start,end:不包含)
          substr(start,length)

          toLocaleUpperCase/toLocaleLowerCase
          toUpperCase/toLowerCase

          字符串填充:
          String.padStart(length: 填充后的字符串长度, value: 要填充的值)
          String.padEnd(length: 填充后的字符串长度, value: 要填充的值)

          判断字符串是否以value值结尾:
          String.startsWith
          String.endsWith(value)

          String.trim: 去除字符串前后空格

          String.repeat()

          String.indexOf(value:要查找的值[,startIndex: 从哪个位置开始查])

          String.replace(oldValue: 要被替换的值,newValue: 要用哪个值替换)

          返回指定下标的字符
          String.charAt(index)

          String.includes()
      */
      let str1 = "hello";
      console.log(new String(str1));
      console.log(str1.split('l'));
      let url = "https://www.baidu.com/s?
rsv_idx=2&hisfilter=1&rsv_dl=fyb_n_homepage";
      let split_r1 = url.split("?")[1].split("&");
```

```
        console.log(split_r1);
        // 获取浏览器地址
        // console.log(window.location.href);
        console.log(str1.repeat(4));
        console.log(str1.replace("ll","aa"));
        console.log(str1.includes("h"));
```

19.操作符和语句

```
<script>
        console.log([] instanceof Array);//true
        console.log([] instanceof Object);//true
        function Person() {

        }
        let p = new Person();
        console.log(p instanceof Array);

        console.log("x" in { x: 100, y: 200 });
        console.log("================================");
        let arr = [20, 50, 10];
        arr.add = "add";
        for (let k in arr) {
            console.log(k);
        }
        console.log("================================");

        let obj = { x: 100, y: 200 }
        for (let val of arr) {
            console.log(val);
        }
        console.log("================================");
        for (let [key, val] of Object.entries(obj)) {
            // console.log(k[0],k[1]);
            console.log(key, val);
        }
        for (let k of Object.keys(obj)) {
            console.log(k);
        }
```

```
        for (let val of Object.values(obj)) {
            console.log(val);
        }

        console.log("================================");

        try {//
            let a = "mya";
            console.log(a);
            console.log(aaaa);
            console.log("try");
        }
        catch (err) {
            // 当try里面的代码有错误时才会执行
            console.log(err);
        } finally {
            // 最终都会执行
            console.log("finally");
        }
        console.log(111);
    </script>
```

20.继承

```
        function Person(name) {
            this.name = name;
            this.say = function () {
                console.log(this.name);
            }
        }
        let p = new Person();
        // 1.通过原型继承
        /* function Student(){

        }
        Student.prototype = Person.prototype;
        let s1 = new Student(); */

        // 2.方法劫持
```

```
        function Student() {
            Person.call(this);
        }
        let s1 = new Student();
        console.log(s1);

        // 3.类的继承
        class myClass {
            constructor(name) {
                this.name = name
            }
            sayName() {
                console.log("sayName");
            }
        }
        class son extends myClass {
            constructor(name, age) {
                super(name);
                this.age = age;
            }
        }
        let o = Object.create(Person.prototype);
        o.add = "add";
        console.log(o);
```

21.ES6

```
// 数组解构赋值
        let [a, b = 22] = [10, null];//当传入的参数严格上是非undefined时，默认值不起作用
        console.log(b); // null
        function fn1([x, y]) {
            console.log(x, y);
        }
        fn1([1, 2]);
        let { y } = { x: 100, y: 200 };
        function fn2({ a, b }) {
            console.log(a, b);
        }
        fn2({ a: 100, b: 200 });
```

```
        function fn3(num1, num2 = 200) {
            console.log(num1, num2);
        }

        function fn4(a, b, ...arr) {
            console.log(a, b, arr);
            console.log(arguments);
            console.log([...arguments]);
            console.log(Array.from(arguments));
        }
        fn4(1, 2, 3, 4, 5)
        console.log("========================");
        console.log(Array.from([10, 20, 30], (v, i) => {
            console.log(v, i);
            return i;
        }));
        console.log(Array.from({ 0: 111, 1: 222, length: 4 }));
        console.log(Array.of(10, 5));
        console.log("========================");
        // 对象的属性名只能是字符串和symbol值
        let s1 = Symbol();
        let s2 = Symbol();
        console.log(s1 == s2);//false
        let s3 = Symbol.for("name");
        let s4 = Symbol.for("name");
        console.log(s3 == s4);//true
        let n1 = 100, n2 = 200;
        let obj1 = {
            n1, n2,
            fn() {

            },
            [s1]: "obj1",
            [s3]: "guang"
        }
        Object.defineProperty(obj1, "age", {
            value: 20,
            writable: true,//控制当前属性值是否可修改
            enumerable: false,//控制当前属性是否可枚举
            configurable: true//控制当前属性是否可删除
```

```
        });
        console.log("修改age前：", obj1);
        obj1.age = 22;
        console.log("修改age后：", obj1);
        // delete obj1.age;
        // console.log("删除age后：",obj1);

        for (const key in obj1) {
            console.log(key);
        }
        // 判断当前对象身上是否拥有某个属性，不能判断原型上的方法
        console.log(obj1.hasOwnProperty("age"));
        console.log("========================");
        // 判断两个值是否相等，===
        console.log(Object.is(10, "10"));//false
        console.log(Object.is(NaN, NaN));//true
        console.log(Object.assign(obj1, { x: 100 }, { y: 100 }));
        console.log({ ...obj1, ...{ a: "mya" } });

```

22.深度复制

```
<script>
        let person = {
            name: 'aaa',
            age: 19
        }

        function copy(obj) {
            let obj1 = {}
            for (let key in obj) {
                obj1[key] = obj[key]
            }
            return obj1;
        }
        let obj1 = copy(person)
        obj1.name = "bbb";
        obj1.age = 20;
        console.log(obj1);
```

```
18
19          // Object.assign()
20          var obj2 = {
21              name: "张三",
22              age: 20,
23              speak: function () {
24                  console.log("我是" + this.name);
25              }
26          };
27
28          var obj3 = Object.assign({}, obj1);
29
30          // 当修改obj2的属性和方法的时候，obj1相应的属性和方法不会改变
31          obj3.name = "李四";
32          console.log(obj2);
33          console.log(obj3);
34      </script>
```

23.CSS水平垂直居中

```
1  元素定宽高：
2      1.定位+calc
3        父元素
4          position: relative;
5        子元素
6          position: absolute;
7          top: calc(50% - 50px);
8          left: calc(50% - 50px);
9
10     2.定位+magin：负值
11         父元素position: relative;
12         子元素position: absolute;
13         top:50%;
14         left:50%;
15         margin: -50px;
16
17 元素不定宽高：
18     3.定位+magin:auto;
19       父元素
```

```
        position: relative;
    子元素
        position: absolute;
        top: 0;
        bottom: 0;
        left: 0;
        right: 0;
        margin: auto;

  4.定位+transform
父元素position: relative;
    子元素position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);

  5.flex布局:
  父元素设置:flex布局,子元素上使用: margin:auto; 居中展示
    1.父元素: display:flex;
      子元素: magin:auto;
    2.display:flex;
      justify-content: center;
      align-items: center;

  6.grid布局
    父元素: display: grid;
    子元素: magin: auto;

    父元素display:grid;
    子元素align-self: center;
    justify-self: center;
   父元素设置:grid栅格布局,align-content: center;justify-content: center; 居中展示
    display: grid;
    align-items: center;
    justify-content: center;

  7.table布局
      父元素
        display: table;
      子元素
```

```
        display: table-cell;
        vertical-align: middle;
        text-align: center;

    父元素
        display: table-cell;
        vertical-align: middle;
        text-align: center;
    子元素
        display: inline-block;

    父元素
        display: table-cell;
        vertical-align: middle;
    子元素
        margin: auto;
```